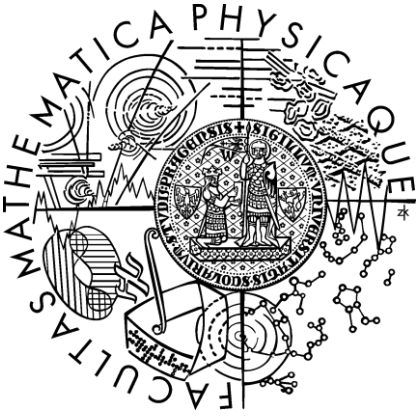
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

# BAKALÁŘSKÁ PRÁCE




Jozef Lelič

## Implementácia API pre vývoj IVA v Minecrafte

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: Mgr. Michal Bída

Studijní program: Informatika
Studijní obor: Programování a softwarové systémy

Praha 2015

Touto cestou by som sa rád poďakoval svojmu vedúcemu Mgr. Michalovi Bídovi za jeho pomoc a za možnosť pracovať na tejto zaujímavej téme.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne ..... podpis

Název práce: Implementácia API pre vývoj IVA v Minecrafte

Autor: Jozef Lelič

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: Mgr. Michal Bída

Abstrakt: Minecraft je počítačová hra umožňující hráčům v 3D světě složeném z bloků stavět různé struktury, lovit zvířata, těžit materiály a celkově měnit její prostředí. Cílem této práce je vytvořit framework umožňující tvorbu virtuálních agentů (botů), kteří jsou schopni hráči pomoci s jednoduchými činnostmi, například tažení či stavba budov. Framewok by měl být snadno použitelný a rozšiřitelný. Součástí práce je také implementace vzorových agentů, kteří dokáží řešit základní úkoly ve hře.

Klíčová slova: inteligentní virtuální agenti, API, Minecraft, vývoj UI

Title: API implementation for IVA development in Minecraft

Author: Jozef Lelič

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Michal Bída

Abstract: Minecraft is a 3D computer game enabling players living in a Minecraft block world to create structures, hunt animals, mine materials and virtually change the shape of their surrounding environment all together. However some of the aspects of the game can become repetitive and boring. The goal of this thesis is to create a framework enabling the player to develop intelligent virtual agents (or bots) that will help the player with the repetitive tasks (such as mining or building). The framework should be easy to use and should allow for extensions. Part of the thesis is to create a set of example agents solving basic tasks in the game.

Keywords: intelligent virtual agents, API, Minecraft, AI development

# Obsah

# Kapitola 1

# Úvod

Populárna počítačová hra Minecraft ponúka vďaka svojej zdanlivej jednoduchosti na prvý pohľad ideálne prostredie pre vývoj virtuálnych agentov. Virtuálnym agentom myslíme agenta existujúceho vo virtuálnom svete. Agent sa môže v tomto svete pohybovať a vykonávať rôzne rozhodnutia v závislosti od jeho stavu. V práci budeme takýchto virtuálnych agentov označovať taktiež pojmom bot.

Hráči môžu často v hre narážať na opakovanie rovnakých činností. Napríklad na stavanie budov potrebujú drevo. Získať drevo je vcelku jednoduché, no hráč musí zvyčajne opustiť miesto stavby aby našiel stromy, odstrániť zo stromov listy, ktoré blokujú prístup apod. V konečnom dôsledku takáto nenáročná a nezábavná činnosť môže zabrať veľa času. Virtuálni agenti by mohli hráčov od týchto činnosti odbremeniť a umožniť im sústrediť sa na zábavnejšie časti hry. V posledných rokoch vzniklo niekoľko prác, ktoré sa o to snažili. Väčšina z nich však pokrýva len malé množstvo herných mechaník a teda nedokáže spoľahlivo nahradiť ľudského hráča.

Ďalšou motiváciou pre túto prácu je sprístupnenie bohatého prostredia Minecraftu pre experimenty v oblasti riadenia virtuálnych agentov. Tým, že Minecraft poskytuje takéto prostredie s radou na riešenie netriviálnych problémov (stavanie veľkých štruktúr, boj s nepriateľmi, ťaženie surovín, chovanie zvierat a pod.) predstavuje ideálnu platformu pre výskum nových prístupov v oblasti umelej inteligencie.

## 1.1 Ciele práce

Cieľom práce je navrhnúť a implementovať framework pre tvorbu umelej inteligencie botov do Minecraftu. Tento framework bude naprogramovaný v jazyku Java a bude za užívateľa riešiť problémy ako sú pripojenie k hre, komunikácia so serverom, reprezentácia herného sveta, správa inventáru a podobne. Bude uchovávať model herného prostredia, ktorý je viditeľný botovi a bude schopný reagovať na rozličné udalosti v ňom. Taktiež poskytne programátorské rozhranie, umožňujúce

jednoduché využívanie herných mechaník, napríklad stavania či ťaženia. Framework bude v konečnom dôsledku simulovať činnosť herného klienta, no kvôli zákazu používania botov na oficánych serveroch sa bude môcť pripojiť len k takým, ktoré to explicitne povoľujú. Architektúra frameworku bude inšpirovaná platformou Pogamut[1], vďaka čomu bude možné v budúcnosti jednoduché pripojenie frameworku na túto platformu.

Ďalším cieľom je implementácia ukážkových botov postavených na tomto frameworku, ktorí budú demonštrovať jeho možnosti.

## 1.2 Štruktúra prace

Druhá kapitola sa venuje hre Minecraft, jej histórií a v krátkosti zhŕňa základné mechaniky hry. V tretej kapitole popisujeme protokol, ktorý hra využíva na komunikáciu po sieti a niektoré problémy, ktoré z neho plynú. Základné informácie o frameworku sú v kapitole štvrtej. V piatej kapitole sa nachádzajú detaily implementácie niektorých dôležitých častí frameworku. Šiesta kapitola sa venuje popisu botov implementovaných v rámci práce. V závere sú diskutované výsledky práce a naznačené možné budúce využitia.

## 1.3 Príbuzné práce

Pre Minecraft existuje celá rada programov, ktoré pomáhajú automatizovať činnosti v hre. Väčšina takýchto programov preberá za hráča kontrolu nad herným klientom simuláciou stláčania kláves a pohybu myši. Týmto prístupom sa však dajú implementovať len veľmi jednoduché činnosti, nakoľko program nemá ako získavať informácie o prostredí. Taktiež by počas behu programu užívateľ v podstate nemohol používať počítač a neumožňuje fungovanie viacerých botov naraz, preto sme sa rozhodli tento spôsob nepoužiť.

Vytvorením programu, ktorý simuluje činnosť herného klienta podobne, ako náš framework, sa zaoberá niekoľko prácí.

TorchBot[4] ponúka užívateľovi možnosť pripojiť k hre bota a ovládať pomocou jednoduchého grafického rozhrania. Toto rozhranie umožňuje vidieť pozície ostatných hráčov vo svete a komunikovať s nimi pomocou herného chatu. Bot vie ukladať štruktúry blokov zo sveta a následne ich zreprodukovať.

SpockBot[5] ponúka možnosť tvorby botov, ktorí dokážu ťažiť, stavať, komunikovať cez chat a pohybovať sa v prostredí. Neponúka síce grafické rozhranie, no umožňuje užívateľom upravovať správanie bota pomocou pluginov.

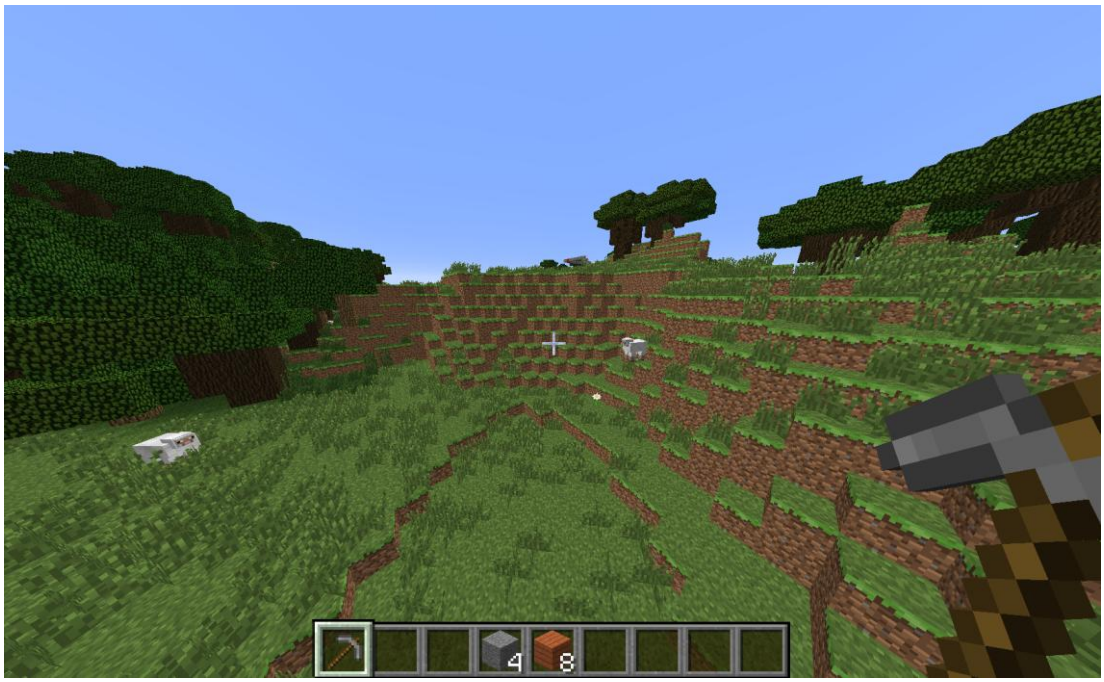
Náš framework sa snaží najmä o podporu čo najväčšieho počtu herných mechaník. Ovláda ich viac ako vyššie spomínané práce.

# Kapitola 2

# Minecraft

## 2.1 O hre

Minecraft je nezávislá sandboxová[1] počítačová hra. Odohráva sa vo svete, ktorý je takmer celý vytvorený z blokov (kociek). Hráči môžu tieto bloky ničiť, ťažiť z nich, ale aj stavať nové. Bloky majú rozličné vlastnosti, v závislosti od materiálu, z ktorého sú vytvorené. Napríklad niektoré sa dajú ničiť ľahko, iné len pomocou špeciálnych nástrojov, ďalšie sa zas používajú na vytváranie (*crafting*) týchto nástrojov. Vo svete sa taktiež nachádzajú rôzne zvieratá a príšery. Tie agresívne môžu byť pre hráča hrozbou, zatiaľ čo neutrálne umožňujú hráčovi získavať potravu potrebnú na prežitie, či už ich chovom alebo lovom. Ďalšou možnosťou zaobstarávania potravy je pestovanie rastlín. Po úmrtí sa hráč môže, podľa módu, oživiť a pokračovať ďalej v hraní. Hra nemá definitívny koniec. Za jej hlavný cieľ sa môže považovať zabitie tzv. Ender draka, najsilnejšej príšery v hre. Drvivá väčšina hráčov ale túto časť hry ignoruje a hrá ju kvôli jej kreatívnym možnostiam.



**Obrázok 1.1: Snímok z prostredia hry Minecraft**

[1] Žáner hry, kde má hráč neobmedzený pohyb v hernom svete a môže si ho rozlične upravovať

## 2.1 História

Hru začal vyvíjať v roku 2009 Švéd Markus "Notch" Persson. V tom istom roku založil spoločnosť Mojang. Prvá oficiálna plná verzia hry pre PC bola hra vydaná v septembri 2011. Minecraft sa už od začiatku stal veľmi populárny. Napríklad na Game Developers Conference 2011 vyhral cenu za inováciu, za najlepšiu novú hru a za najlepšiu hru na stiahnutie. Vďaka veľkému úspechu boli vytvorené edície hry aj pre iné platformy: Android, iOS, Windows Phone, PlayStation 3, 4 a Vita, Xbox 360, dokonca aj Raspberry Pi. V septembri roku 2014 spoločnosť Mojang aj s právami na Minecraft odkúpil Microsoft.

## 2.2 Módy

V Minecrafte existuje niekoľko módov, ktoré udávajú hráčove možnosti. Rôzni hráči v rovnakej hre môžu hrať v rôznych módoch.

**Survival**

Toto je základný mód. Všetky potrebné materiály a nástroje si v ňom musí hráč zaobstarávať sám. Hráč môže zomrieť po útoku príšer, páde z veľkej výšky alebo od hladu. Po úmrtí hráčovi vypadnú všetky predmety a objaví sa na mieste, kde začínal. Hráč teda môže získať svoje veci späť, ak sa vráti na miesto svojho úmrtia. Framework je vytvorený primárne pre tento mód.

**Creative**

V tomto móde má hráč dostupné všetky predmety v hre, bez toho aby ich musel získavať. Hráč je nesmrteľný a môže lietať. Nepotrebuje žiadne nástroje na ťažbu blokov, všetky dokáže okamžite zničiť vlastnými rukami. Mód je určený pre hráčov, ktorí sa chcú zamerať najmä na stavanie.

**Hardcore**

Rovnaký ako survival mód, avšak hráč môže umrieť len raz. Hra preňho potom končí.

## 2.3 Obtiažnosti

Hra obsahuje 4 obtiažnosti, ktoré určujú silu a inteligenciu príšer v hre. Na najľahšej obtiažnosti (peaceful) sú z hry všetky tieto príšery odstránené. Hráč nemôže jesť jedlo, ale zároveň ani netrpí hladom. Ak sa hráč aj napriek tomu zraní, napríklad pádom z výšky, jeho zdravie sa postupne naspäť obnoví. Táto obtiažnosť je ideálna pre bota, preto je framework určený pre ňu.

Ďalšie obtiažnosti sú easy, medium a hard. V nich už hráč narazí na nepriateľské príšery. Kým v easy sú tieto príšery pomerne slabé, v hard už niektoré dokážu napríklad vyraziť dvere. Hráč musí pravidelne jesť, inak mu bude ubúdať život do určitej hladiny, prípadne na najťažšej obtiažnosti môže od hladu zomrieť.

# Kapitola 3

# Vlastnosti protokolu

Na komunikáciu po sieti hra využíva binárny protokol. Server prijíma spojenia klientov a komunikuje s nimi pomocou paketov. Odosiela ich každých 50 milisekúnd. Dodržanie tejto frekvencie sa od klientov nutne nevyžaduje. Ich formát je popísaný nižšie. Keďže hra je naprogramovaná v jazyku Java, dátové typy z neho sú zhodné s tými, použitými v protokole. Malá výnimka nastáva pri type *String*. Ten používa kódovanie UTF-8 a na začiatku je vždy číslo udávajúce jeho dĺžku. Okrem toho protokol definuje ešte jeden nový dátový typ - *Varint*.

## 3.1 Varint

Varint slúži na uloženie hodnoty celého čísla, pomocou jedného alebo viacerých bytov. Jeho dôležitou vlastnosťou je, že na menšie čísla spotrebuje menej bytov ako na väčšie. Každý byte vo Varint okrem posledného má nastavený najvýznamnejší bit (most significant bit). Ten teda indikuje, či ešte nasledujú po danom byte ďalšie. Ostatných 7 bitov už potom obsahuje údaj o hodnote daného čísla, pričom najmenej významná skupina týchto bitov je prvá. Číslo 6 bude teda vyzerať takto:

0000 0110 → 0b 000 0110 = 6

Číslo 300 už je o niečo väčšie, zaberá teda aj viac bytov:

1010 1100 0000 0010 → 0b 000 0010 010 110 = 300

## 3.2 Pakety

Jednotlivé správy, ktorými komunikujú v Minecrafte server a klient sú rozdelené do paketov. V tomto kontexte pod pojmom paket rozumieme prenášanú postupnosť bytov popisujúcu práve jeden typ správy z protokolu. Na začiatku paketu je jeho dĺžka v bytoch. Samotný údaj o dĺžke sa do nej nezarátava. Po nej nasleduje

identifikačné číslo daného paketu, určujúce typ posielanej správy. Formát zvyšku paketu už záleží od konkrétneho typu.

K protokolu neexistuje verejne dostupná oficiálna špecifikácia, no fanúšikom hry sa ho podarilo zdokumentovať takmer celý[6]. Celkovo protokol definuje 80 typov paketov pre komunikáciu smerom zo servera ku klientovi a 32 typov pre opačný smer. Pre lepšiu predstavu uvedieme popis niektorých dôležitých:

***Keep Alive***

Náklad: *Keep Alive Id*

Tento typ paketu má variáciu pre oba smery. Používa sa na uistenie, či je klient pripojený k serveru. V náklade paketu sa nachádza náhodne vygenerované číslo. Klient musí toto isté číslo poslať serveru v pakete *Keep Alive* naspäť, inak dôjde k jeho odpojeniu.

***Player Position and Look***

Náklad: *pozícia hráča vo svete, uhol otočenia hlavy hráča*

Tento paket sa taktiež používa v oboch smeroch. Klient ho posiela pri zmene svojej pozície a otočenia hlavy. Server posiela tento paket v dvoch prípadoch. Pri pripojení nového hráča mu ním oznamuje jeho začiatočnú pozíciu. Ak hráč spraví nekorektný pohyb, server mu dá týmto paketom vedieť, že zmena pozície nebola akceptovaná.

Pokiaľ klient len stojí na mieste a otáča hlavou, môže použiť paket *Player Look*, ktorý popisuje len zmenu otočenia hlavy. Analogicky pre prípad kedy hráč mení pozíciu no neotáča hlavou existuje paket *Player Position*,

***Spawn Player***

Náklad: *údaje o novom hráčovi – id, meno, pozícia apod.*

Tento paket sa používa iba v smere server → klient. Server ním informuje klienta o hráčovi v hre, ktorého doteraz klient nepozná. Neposiela sa však pri pripojení tohto hráča na server, ale až keď sa tento hráč dostane do viditeľnej vzdialenosti klienta. V dôsledku to teda znamená, že pokiaľ nemá klient hráčov vo viditeľnej vzdialenosti, nepozná ich pozíciu.

***Set Slot***

Náklad: *číslo slotu(políčka), údaje o novom predmete*

Tento paket sa používa iba v smere server → klient. Server ním informuje klienta, že sa v určitom políčku v jeho inventári zmenil typ predmetu. Konkrétne v situácii, kedy hráč daný predmet zdvihne zo zeme. Pri presúvaní predmetov v inventári a craftení si musí klient tieto zmeny riadiť sám. Inventár je popísaný v časti 4.9.

***Click Window***

Náklad: *číslo slotu(políčka, údaje o predmete na políčku), číslo akcie*

Tento paket sa používa iba v smere klient → server. Klient ním informuje server, že hráč v inventári klikol na určité políčko. Klient musí však zároveň popísať predmet, ktorý sa na tomto políčku nachádza. Klient aj server teda uchovávajú stav inventára klienta a týmto údajom server overuje, že oba stavy sú rovnaké.

Každá takáto akcia (kliknutie) má vlastné identifikačné číslo. Číslo môže klient zvoliť ľubovoľné, dve akcie v rozpätí niekoľkých sekúnd však nemôžu mať toto číslo rovnaké.

***Confirm Transaction***

Náklad: *číslo akcie, stav potvrdenia akcie (true/false)*

Tento paket sa používa iba v smere server → klient. Server ním informuje klienta o tom, či bola akcia vykonaná hráčom v inventári korektná. Nekorektné akcie nastávajú, keď informácia o predmete, na ktorý hráč klikol sa nezhoduje s tou uloženou na serveri. V tom prípade server pošle celý jeho stav inventára klientovi.

## 3.3 Herná fyzika

Klienti v Minecrafte posielajú údaje o svojej pozícii serveru. Ten skontroluje, či táto pozícia nekoliduje s nejakým hmotným blokom alebo či nie je príliš ďaleko od predchádzajúcej. Pokiaľ nastal niektorý z týchto prípadov, server neuzná zmenu pozície a pošle naspäť klientovi jeho poslednú korektnú pozíciu. To má niekoľko

závažných následkov. Každý klient, a teda aj náš framework, je sám zodpovedný za svoju gravitáciu. Môžeme ju samozrejme odignorovať, no dedikovaný server odpojí hráčov, ktorí sa vznášajú vo vzduchu niekoľko sekúnd. Lokálny server s tým problém nemá.

Na detekciu kolízii sa v Minecrafte používa spôsob AABB (Axis-Aligned Bounding Box)[7]. Každý hráč je v ňom reprezentovaný kvádrom so štvorcovou podstavou, pričom hrany kvádra sú rovnobežné s jednotlivými osami sveta. Kým tento spôsob nie je úplne presný, je veľmi jednoduchý a vzhľadom na vlastnosti sveta v Minecrafte postačujúci.

# Kapitola 4

# Popis frameworku

## 4.1 Základné vlastnosti

Framework bol naprogramovaný v jazyku Java. Okrem typických výhod, ktoré tento jazyk ponúka, napr. prenositeľnosť, nám výber uľahčila skutočnosť, že aj samotná hra Minecraft bola naprogramovaná v tomto jazyku. Ďalším dôvodom je neskoršia možnosť pripojenia frameworku na platformu Pogamut.

Nakoľko každých niekoľko mesiacov vychádza nová verzia Minecraftu, rozhodli sme sa vytvoriť framework pre verziu 1.7.5, ktorá bola aktuálna v čase začiatku práce. Architektúra frameworku však bola navrhnutá tak, aby bolo jednoduché reagovať na zmeny v protokole. V budúcnosti by teda nemalo byť ťažké upraviť framework na fungovanie s novou aktuálnou verziou. Navyše hra pri spustení umožňuje zvoliť užívateľovi ľubovoľnú verziu.

Používanie botov je na oficiálnych serveroch zakázané, keďže môže poskytnúť jeho užívateľovi veľkú výhodu nad ostatými hráčmi. Framework sa vie preto pripojiť len na lokálne servery (vytvorené priamo herným klientom) a na dedikované servery s vypnutím nastavením „*online-mode“*. Takéto servery neoverujú identitu klientov.

## 4.2 Podporované herné mechaniky

Celkovo framework podporuje nasledujúce herné mechaniky:

- Reprezentácia herného sveta
  - uchovávanie informácii o každom bloku v blízkosti bota
  - uchovávanie pozícii dôležitých blokov
- Simulácia gravitácie
- Základne pohyby
  - pohyb po zemi, skákanie, otáčanie sa a otváranie dverí
- Navigácia v prostredí

- Informácie o pripojených hráčoch
    - meno, pozícia, doba odozvy

- Stavanie
    - od umiestňovania jednotlivých blokov až po automatickú stavbu veľkých štruktúr
    - možnosť načítavania štruktúr na postavenie z textových súborov

- Ťaženie
    - možnosť automatického najvhodnejšieho nástroja ťažbu daných blokov

- Lokalizácia predmetov na zemi
    - bot dokáže nájsť predmety, ktoré hráči vyhodia, prípadne ktoré vzniknú ťažením

- Správa inventáru
    - bot vie, aké predmety má v inventári
    - bot si vie presúvať predmety v inventári a vyhadzovať ich z neho

- Jednoduchý automatický crafting
    - keď bot dostane drevo, automaticky z neho vycraftí drevené dosky, ktoré sa používajú na stavbu budov

- Chat
    - bot dokáže odosielať a príjmať správy z herného chatu

- Reakcia na udalosti
    - bot dokáže zachytávať a reagovať na udalosti v hre
    - pripojenie nového hráča k hre, odpojenie hráča, smrť hráča

Implementácia niektorých dôležitých mechaník je popísaná v Kapitole 5.

## 4.3 Architektúra frameworku

Keďže server odosiela správy každých 50 milisekúnd, framework sa snaží toto správanie napodobniť. V hlavnom cykle, implementovanom v metóde *loop()* triedy *MinecrafBotHandler*, načítava prijaté pakety. Tie následne oddelí od seba, a postupne ich spracuje. Na spracovanie jednotlivých paketov sa používajú obsluhovači, ktorí implementujú rozhranie *IInPacketHandler*. Každý podporovaný typ paketu má priradený svoj obsluhovač. Nepodporované pakety sa odignorujú. Obsluhovači po spracovaní prijatých paketov predávajú informácie z nich jednotlivým komponentom frameworku, ktorých činnosť na nich závisí. To umožňuje jednoduché nahradenie niektorých komponent frameworku, bez znalosti

protokolu. Taktiež umožňuje jednoduchšie reagovať na zmeny protokolu, nakoľko stačí upraviť obsluhovačov.

Pre riadenie každej činnosti bota existuje trieda, ktorá ju implementuje. Tieto triedy prijímajú informácie o svete od obsluhovačov paketov. Na ich základe menia svoj vnútorný stav a vyvolávajú herné udalosti. Užívateľovi sú prístupné cez príslušné rozhrania, ktoré mu takto umožňujú zadávať botovi príkazy. Na konci hlavného cyklu prebieha aktualizácia stavov týchto tried a prípadne odosielanie správ o novom stave na server.

Užívateľ môže riadiť chovanie bota pomocou obsluhy herných udalostí. Pri vyvolaní každej udalosti je zavolaná príslušná metóda triedy *MinecraftBotHandler*. Základnou udalosťou je *update*. Nastáva pri každom štvrtom prechode cyklom, čiže každých 200 milisekúnd. Zoznam všetkých udalostí a ich popis sa nachádza v priloženej užívateľskej dokumentácii.

# Kapitola 5

# Implementácia frameworku

## 5.1 Komunikácia so serverom

Pre správne fungovanie musí byť framework schopný spracovať serverom odoslané správy a vedieť na ne korektne reagovať. Keďže dĺžka paketu a typ správy, ktorý popisuje sa nachádza na jeho začiatku, môžeme jednoducho ignorovať správy nepodstatné pre náš framework. Jedná sa napríklad o rozličné vizuálne a zvukové efekty nevplývajúce na fungovanie bota. Konkrétne sa o komunikáciu so serverom starajú 3 triedy, ktoré sú potomkami stream tried z jazyka Java.

***MinecraftInputStream***

Je potomkom triedy *InputStream*. Pracuje priamo s prúdom dát zo servera. Jeho úlohou je zistiť dĺžku a typ paketu na vstupe. Následne môže načítať obsah celého paketu do poľa a vytvoriť nad ním *MinecraftDataInputStream*.

***MinecraftDataInputStream***

Je potomkom triedy *DataInputStream*. Pracuje s dátami z jedného paketu. V podstate sa jedná o *DataInputStream* vytvorený nad *ByteArrayInputStream*, ktorý ako zdroj využíva pole s dátami z paketu. Naviac má však definované ešte dve metódy. Na čítanie typu *Varint* vo formáte definovanom protokol metóda *readVarint()*, obdobne *readString()* pre formát *String*.

Oddelenie jedného paketu od celého prúdu zvyšuje bezpečnosť, nakoľko nemôže nejaká časť programu omylom prečítať dáta z iného paketu a narušiť tak integritu celého prúdu.

***MinecraftOutputStream***

Je potomkom triedy *OutputStream*. Na rozdiel od nej však nezapisuje dáta priamo ale do poľa. Až pri zavolaní metódy *closePacket()* sa spočíta veľkosť uložených dát a spolu s obsahom poľa zapíše do prúdu. To uľahčuje zachovanie tvaru paketov podľa protokolu, keďže ich veľkosť sa počíta automaticky.

## 5.2 Reprezentácia sveta

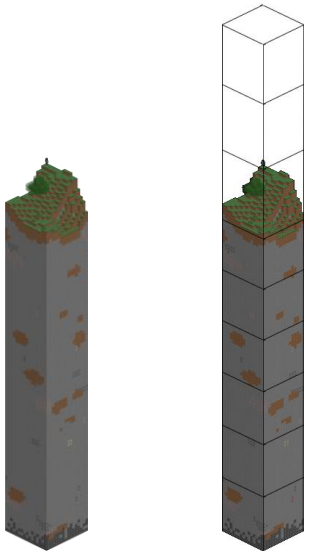
Svet v Minecrafte sa skladá z blokov. O každom bloku hra ukladá dve informácie:

- ID
    - určuje typ bloku
    - závisia od neho fyzikálne vlastnosti bloku – priechodnosť, priehľadnosť
    - napr. drevo, kameň atď.
- metadata
    - obsahujú ju iba niektoré typy blokov
    - väčšinou udáva len vizuálne vlastnosti
    - napr. farba dreva, smer natočenia bloku

Pri pripojení na server, posiela klient serveru svoje lokálne nastavanie. Medzi ne patrí aj hodnota *render distance*. Meria sa v tzv. chunkoch. Chunk je štruktúra zložená z 16 x 16 x 256 blokov. Jej hodnota udáva vzdialenosť blokov od hráča, ktoré klient ešte vykreslí. Ak pripojenie prebehlo úspešne server pošle klientovi informácie o každom bloku, ktorý sa nachádza vo vzdialenosti menšej, ako je jeho hodnota *render distance.*

Informácie o blokoch sa môžu posielať tromi rôznymi spôsobmi. Pri vykopaní alebo pokladaní bloku hráčom sa posielajú informácie o tomto jednom bloku. Pri výbuchoch, kedy sa zničí viacero blokov spolu sa posiela zoznam týchto blokov. Pri pripojení, teleportovaní alebo pohybe hráča po mape sa neposielajú informácie o jednotlivých nových blokoch ale o celých chunkoch.

Keďže väčšinu sveta v Minecrafte zaberajú prázdne bloky, môžeme túto skutočnosť využiť na ušetrenie pamäte. Chunk si rozdelíme na menšie časti o veľkosti 16 x 16 x 16 blokov. Ak pri čítaní informácii o chunku, zistíme, že nejaká takáto časť neobsahuje žiaden „hmotný“ blok, nemusíme ju uložiť informácie o žiadnom bloku v nej.



**Obrázok 5.1: Príklad rozdelenia chunku na kocky**

Pri reprezentácii prostredia môžeme ešte ďalej ušetriť pamäť s využitím hlavnej myšlienky návrhového vzoru flyweight. Tento vzor sa používa v situácii, kedy potrebujeme reprezentovať veľké množstvo objektov, z ktorých väčšina má rovnaký vnútorný stav. Vo svete Minecraftu sa nachádza veľké množstvo blokov (rádovo potrebujeme uchovávať informácie o miliónoch), pričom väčšina z nich sú rovnaké (hlina, kameň, drevo, zvyčajne nie viac ako 100 druhov). Miesto vytvárania nového objektu pre každý jeden blok preto vytvoríme objekt len pre každý druh bloku. O ich správu sa stará tzv. flyweight factory (vo frameworku implentovaný triedou *BlockFactory*). Flyweight factory na požiadanie vráti objekt odpovedajúci danému druhu. Musí teda uchovávať objekty pre používané druhy blokov a pri požiadavku o nový druh, vytvoriť a uložiť odpovedajúci objekt. Podľa návrhového vzoru Flyweight by sa zároveň malo dbať na odstraňovanie objektov, ktoré sa už nepoužívajú. V Minecrafte však nemá zmysel túto funkcionalitu implementovať. Aby hráč úplne z mapy odstránil nejaký druh bloku je totiž extrémne zriedkavé. Hrou vopred vygenerované druhy blokov sa nachádzajú v prostredí vo veľkom množstve a bloky vytvorené hráčmi (napr. pracovný stôl) väčšinou hráči už neničia. Ušetrená pamäť by bola teda príliš malá vzhľadom na overhead potrebný na realizáciu tejto funkcionality.

## 5.3 Fyzika a navigácia

Základné funkcie spojené s pohybom bota sú *implementované* pomocou triedy *Locomotion*. Ta obsahuje metódy umožňujúce jednoduchý pohyb bota do rôznych smerov, otáčanie, skákanie či otváranie dverí. Stará sa taktiež o simuláciu gravitácie. Keďže si hra nevynucuje konkrétny spôsob implementovania gravitácie, bot padá zemi pomalšie ako klasický hráč. Vďaka tomu sa zraňuje menej pri páde z veľkých výšok.

Kým detekciu kolízií nie je v podstate nutné implementovať, nakoľko server nedovolí zmeniť pozíciu hráča tak, aby ku kolízii došlo, nie je to úplne vhodný prístup. Server pri kolízii opraví pozíciu hráča len na poslednú korektnú, čo by robilo problém, ak by sme sa naň spoliehali pri simulácii gravitácie. Ak by sa bot totiž pokúsil „spadnúť“ o väčšiu vzdialenosť, než sa nachádza nad zemou, server by ho vrátil naspäť a takto by sa nikdy nedotkol zeme. Preto framework vypočítava budúce pozície bota pri padaní.

Pri horizontálnom pohybe framework kolízie nerieši. Problém, ktorý by nastával pri padaní je v tomto prípade oveľa menej viditeľný. Bot sa zvyčajne pohybuje rýchlosťou menšou ako je polovica dĺžky hrany bloku za jeden tik hry. Takže pokiaľ dôjde k horizontálnej kolízii, bot ostane stáť od prekážky menej ako je polovica bloku, čo nie je vcelku nepatrná vzdialenosť.

O navigáciu sa stará trieda *Navigation*. Používa algoritmus A*[8]. Ako uzly na hľadanie cesty používa algoritmus nehmotné bloky. Použitá heuristická funkcia je vzdušná vzdialenosť medzi danými uzlami, resp. blokmi. Bot dokáže využívať pri navigácii skákanie. To má však o niečo horšie ohodnotenie ako normálne chodenie a teda bot použije cestu obsahujúcu skoky, len ak je kratšia ako cesta bez skokov. Obdobne je to s blokmi listov. Tieto bloky v Minecrafte reprezentujú stromy a je ich teda vo svete pomerne dosť, no dajú sa vcelku rýchlo odstrániť. Bot ich preto považuje za priechodné bloky a keď na nich pri ceste narazí jednoducho ich odstráni. Analogicky bot považuje dvere za priechodné bloky a pri ceste si ich otvorí pokiaľ sú zatvorené.

Kvôli veľkému množstvu blokov môže výpočet ideálnej cesty zabrať dlhší čas, zvlášť keď hľadaná cesta neexistuje. Pokiaľ sa to stane v nevhodnom čase, môže server dokonca bota odpojiť, keďže nereaguje na jeho správy. Preto sa tento výpočet prevádza v samostatnom vlákne.

## 5.4 Ťaženie

Ťaženie je vo frameworku vcelku priamočiare. Použitím metódy *mine()* triedy *Mining* s parametrami súradníc daného bloku sa bot pokúsi daný blok vyťažiť. Metóda sa môže volať niekoľkokrát za sebou, skôr než bot dokončí ťaženie. Framework si uloží jednotlivé súradnice blokov a postupne sa ich pokúsi vyťažiť, keď nebude práve ťažiť niečo iné. Pokiaľ bot nedokáže vyťažiť blok do piatich sekúnd ťaženie zlyhá a nastane udalosť *onMiningFail,* v opačnom prípade *onMiningSuccess*. Ak sa nachádza bot príliš ďaleko od bloku, ktorý chce vyťažiť, znovu nastane *onOutOfRangeToMine*.

Pre vzácnejšie bloky si framework uchováva zoznam ich pozícií vo svete. To umožňuje rýchlo nájsť potrebné suroviny. Zoznam surovín, ktorých pozície sa ukladajú je možné upraviť pri spustení bota.

Bot sa implicitne snaží na každý blok, ktorý má vyťažiť, použiť vhodný nástroj, ak ho má v inventári. Toto chovanie je možné zakázať pre prípad kedy je vhodnejšie tieto nástroje šetriť.

## 5.5 Stavanie

### 5.5.1 Základné stavanie

Stavanie patrí medzi jednu zo základných mechaník v Minecrafte. Framework obsahuje niekoľko spôsobov ako ho realizovať. Všetky využívajú abstraktnú triedu *Buildable*, ktorej metódy slúžia na postavenie základných štruktúr. Každá z nich má ako prvý parameter *id* použitého bloku. Zvyšné parametre udávajú pozíciu štruktúry. Ich význam sa líši v závislosti od konkrétnej metódy.

- *block*
    - postaví jeden blok
    - pozícia je udaná súradnicami bloku

- *collumn*
    - pozícia je udaná súradnicami najnižšieho bloku a výškou daného stĺpu v blokoch

- *wall*
    - postaví vertikálnu stenu
    - pozícia je udaná súradnicami dvoch najvzdialenejších blokov v stene, tj. najnižším blokom na jednom konci steny a najvyšším blokom na druhom konci
- *platform*
    - postaví štvorcovú platformu s hrúbkou 1 blok
    - pozícia je udaná súradnicami dvoch protiľahlých blokov, ak sa nachádzajú v rôznych výškach, použije sa výška prvého bloku

**Obrázok 5.2: Základné štruktúry (bloky udávajúce pozíciu sú tmavo označené)**

Pre okamžité postavenie týchto základných štruktúr sa jednoducho zavolajú ich metódy na triede Building. Ak je žiaduce od botov, aby opakovali stavanie nejakej zložitejšej štruktúry, je možné použiť triedu *Structure* popísanú v ďalšej časti.

### 5.5.2 Štruktúry

Medzi jednu z menej obľúbených častí hry patrí opakované stavanie, napríklad domov pri stavaní vlastnej dediny. Framework umožňuje túto úlohu užívateľovi zjednodušiť pomocou triedy *Structure*.

*Structure* slúži na reprezentáciu zložitejšej štruktúry. Je potomkom abstraktnej triedy *Buildable*, takže sa ňou pracuje rovnako ako pri priamom stavaní. Rozdiel spočíva v tom, že pri volaní týchto metód bot nepostaví volaný typ základnej

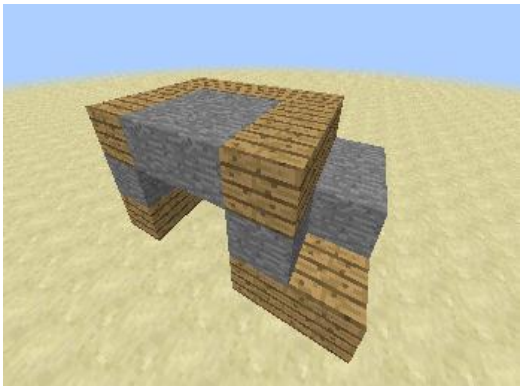
štruktúry automaticky, ale ju vloží do štruktúry zložitej. *Structure* používa vlastnú súradnicovú sústavu. Tá ma stred v bode (0,0,0) a jednotlivé súradnice nemôžu byť záporné. Trieda taktiež obsahuje metódy ako *getFlippedX()* a *getRotatedClockwise()*, ktoré vracajú danú štruktúru rôzne otočenú.

Štruktúry je taktiež možné načítať z textového súboru pomocou statickej metódy *Structure.fromFile(path)*, kde parameter *path* je cesta k danému súboru. Pokiaľ je súbor korektný, vráti metóda štruktúru v ňom popísanú. Súbor sa skladá z dvoch častí. V prvej sú popísané použité typy blokov v štruktúre a znaky, ktoré ich reprezentujú. Medzery sú vyhradené pre prázdne bloky (resp. vzduch). Príklad:

```
x=Wood
o=Glass
s=Stone
```

Každá dvojica sa musí nachádzať na samostatnom riadku. Nasleduje popis jednotlivých úrovní. Každá úroveň začína riadkom s jediným znakom "-". Na nasledujúcich riadkoch sa nachádzajú jednotlivé znaky (definované v prvej časti) odpovedajúce blokom v štruktúre. Príklad:

```
-
xxxxx
x   x
x  xx
-
sooos
s   s
s  s
-
xxxx
xssx
xssx
```



**Obrázok 5.3 Štruktúra z textového súboru postavená botom**

### 5.5.3 Pokročilé stavanie

Kým užívateľsky definované štruktúry je možné postaviť pomocou triedy *Building*, môžu byť veľmi veľké. Kvôli tomu ich bot nebude vedieť postaviť naraz, ale bude sa musieť postupne premiesťovať. Keďže cieľom týchto štruktúr bolo ich opakované použitie, existuje vo frameworku na ich postavenie trieda *AdvancedBuilding*. Tá slúži ako fasáda (v zmysle návrhového vzoru *facade*), ktorá spája dohromady stavanie, výber poradia postavených blokov a navigáciu. Užívateľovi teda stačí zadať štruktúru a pozíciu, kde ju chce postaviť. Pri pokročilom stavaní sa bot snaží stavať štruktúru zdola po jednotlivých poschodiach. To mu pomôže zaistiť, aby sa pri stavaní vedel dostať na vyššie poschodia.

## 5.6 Inventár

Inventár slúži na ukladanie rozličných surovín či nástrojov a následnú prácu s nimi. V hre je reprezentovaný 45 políčkami. 38 z toho slúži na uchovávanie predmetov, 5 na craftenie a 4 reprezentujú výzbroj hráča. Niektoré suroviny je možné položiť na jedno políčko pokope pokiaľ sú rovnakého typu. V závislosti od suroviny ich môže byť na jednom políčku až 64. Nástroje môžu byť na políčku len po jednom. Nástroj, ktorý hráč používa na ťaženie a bloky, ktoré chce do sveta položiť, musí držať v ruke. V ruke môže držať iba predmety nachádzajúce sa na spodných políčkach. Medzi týmito políčkami si potom môže vyberať, ktoré chce použiť.


**Obrázok 5.4: Rozloženie inventáru v Minecrafte**

V Minecrafte je nutné aby si klient uchovával stav svojho inventáru. Tento stav je zároveň uchovaný na serveri. Práca s inventárom je v protokole definovaná cez jednotlivé kliknutia. Vždy keď hráč klikne na políčko v inventári, pošle na server správu kam klikol, čo sa na tom políčku nachádza a v akom množstve. Ten ju môže potvrdiť alebo zamietnuť. Na rozdiel od posielania pozície, v tomto prípade server musí každý klik v inventári schváliť explicitne. Pri častom presúvaní predmetov, najmä za účelom craftenia, môže vznikať veľké množstvo obskúrnych okrajových situácii. Ak ich framework neodsimuluje správne a dôjde k chybe nastávajú problémy pokiaľ je naplánovaných niekoľko akcií vopred. Preto sme sa rozhodli implementovať crafting len v obmedzenej miere. Vždy keď bot zdvihne blok dreva, automaticky z nich vycraftí dosky, ktoré sa používajú na stavanie. Väčšina craftenia, ktoré musí hráč v Minecrafte robiť je totiž práve toto vytváranie dosiek. Implementácia plnohodnotného craftingu ostáva predmetom budúcich prác.

Framework umožňuje užívateľovi zistiť jedine aké predmety má v inventári. Od ich presného rozloženia v ňom je odtienený. Nemôže teda ani sám priamo vyberať, ktorý predmet má bot v držať v ruke. Môže len nepriamo pri stavaní a ťažení, kedy framework automatický zvolí vhodný predmet na danú činnosť podľa toho, akú akciu si užívateľ želal vykonať.

Práca s inventárom je vo frameworku implementovaná cez 3 rozhrania.

- *IInventoryStorage*
    - slúži na spracovanie správ od serveru, napr. potvrdenie kliknutia

- *IInventoryHandler*
    - slúži na presúvanie predmetov v rámci inventáru, umožňuje presunúť do ruky potrebný predmet, napr. krompáč ak bot plánuje ťažiť
    - nie je prístupný užívateľovi
    - príma požiadavky na predmet alebo typ predmetu, ktorý má presunúť do ruky bota, každému požiadavku priradí číslo
    - metóda *isConfirmed(x)*, vracia informáciu, či bola požiadavka s číslom *x* úspešne dokončená

- *IInventoryView*
    - poskytuje informácie, ktoré predmety sú v inventári a ich počet
    - ako jediné z týchto troch rozhraní je prístupné užívateľovi

Framework spravuje inventár pomocou troch druhov akcií. Presunutie predmetu do ruky, craftenie drevených dosiek a vyhadzovanie predmetov. Akcie sa skladajú z niekoľkých kliknutí. Napríklad pri presúvaní predmetu je potrebné kliknúť na požadovaný predmet, následne na políčko reprezentujúce ruku a nakoniec znova na pôvodné miesto predmetu. Po každom kliku je nutné čakať na potvrdenie od serveru. Z týchto dôvodov nie je možné vykonávať viac akcií paralelne. Požiadavky na jednotlivé akcie je teda nutné ukladať do fronty.

Výnimku tvorí akcia presúvania. Bot potrebuje väčšinou okamžite vedieť, či ju bude možné vykonať (teda či sa nachádza v inventári potrebný predmet), aby mohol využiť predmet na stavanie alebo ťaženie. Ak by bola požiadavka na presunutie predmetu do ruky vo fronte za požiadavkou na vyhodenie tohto predmetu, mohol by byť predmet vyhodený zatiaľ čo by bot ostal čakať na jeho presunutie.

Užívateľ nemôže presúvať predmety v inventári priamo. Pri činnostiach, ktoré vyžadujú presun určitých predmetov do ruky sa framework sám rozhoduje, ktorý konkrétny predmet sa použije. Pre každú činnosť existuje práve jeden vhodný typ nástroju, takže dávať užívateľovi možnosť nástroje meniť priamo nie je nutné. Vďaka tomu taktiež potom nemusíme riešiť problémy so synchronizáciou akcií presúvania predmetov, ktoré by užívateľ mohol spôsobiť.

Akcia vyhadzovania predmetov je pre jednoduchosť implementovaná pomocou zatvárania okna inventáru. Miesto viacerých kliknutí, bot jednoducho po jednom kliknutí na predmet zatvorí okno inventára, čo ma v Minecrafte za následok vyhodenia kliknutého predmetu.

# Kapitola 6

# Príklady použitia

V tejto kapitole popisujeme niekoľko ukážkových botov, implementovaných za pomoci nášho frameworku. Hráči môžu ovládať botov pomocou príkazov v hernom chate.

## 6.1 Empty Bot

Tento bot nerobí v hre nič. Môže sa hodiť pre testovanie pripojenia na server, prípadne ako kostra pre tvorbu nového bota.

## 6.2 Responsive Bot

Responsive bot ovláda väčšinu činností implementovaných v frameworku. Vie sa na príkazy hráča pohybovať, skákať, navigovať či vyťažiť blok. Vie oznámiť hráčovi svoju aj jeho pozíciu. Dokáže odpovedať na otázky o typoch blokov vo svete či o počte predmetov vo svojom inventári. Slúži na rýchlu demonštráciu implementovaných herných mechaník.

## 6.3 Lumberjack

Lumberjack má za úlohu ťažiť pre hráča drevo. Hľadá najbližšie stromy, ktoré vyrúbe a pozbiera z nich spadnuté drevo. Pokiaľ sa nedokáže dostať k nejakému stromu kvôli prekážkam v prostredí, bude strom ignorovať. Je mu možné nastaviť cieľ, koľko dreva ma vyťažiť. Po splnení cieľa môže všetko vyťažene drevo odovzdať hráčovi.

## 6.4 Builder

Cieľom tohto bota je stavanie štruktúr, ktoré užívateľ definuje v textových súboroch. Po zadaní príkazu bot príde na pozíciu daného hráča a začne na nej stavať zvolenú štruktúru. Štruktúru môže podľa želania postaviť otočenú inak, než bola definovaná v súbore.

## 6.5 Cleaner

Cleaner prenasleduje hráča a zbiera všetky predmety, ktoré v jeho okolí spadnú na zem. Umožňuje tak hráčovi venovať sa ťaženiu a nestarať sa o zbieranie predmetov zo zeme. Hráč môže botovi kedykoľvek prikázať, aby mu vyzbierané veci dal.

## 6.6 Commentator

Komentuje základné udalosti hre. Víta hráčov pri pripojení na server, lúči sa po odpojení. Ďakuje za každý predmet, ktorý mu hráči dajú. Smúti za zabitými hráčmi a prejavuje nevôľu keď zabíjajú jeho.

# Kapitola 7

# Záver

## 7.1 Výsledky

V rámci práce bol vytvorený framework, ktorý výrazne uľahčuje tvorbu botov pre hru Minecraft. Odbremeňuje užívateľa od riešenia problémov s pripojením k serveru a interpretáciou sieťového protokolu, čím mu umožňuje sústrediť sa na programovanie umelej inteligencie. Framework podporuje väčšinu základných herných mechaník. Pri implementácii craftingu sme sa kvôli jeho zložitosti uskromnili len so zjednodušenou verziou. Architektúra frameworku umožňuje jeho jednoduché rozširovanie bez nutnej znalosti sieťového protokolu hry a teda ponúka mnohé možnosti pre budúce práce.

V práci bolo taktiež vytvorených niekoľko ukážkových botov. Na týchto botoch je demonštrované použitie frameworku k plneniu základných úloh v hre.

## 7.2 Možné budúce rozšírenia

Vzhľadom na rozsah hry sa nabádajú mnohé možnosti rozšírenia frameworku. Medzi mechaniky, ktoré neboli zatiaľ implementované patria napríklad plnohodnotný crafting a boj s nepriateľmi. Hernú fyziku je možné vylepšiť pridaním podpory pohybu vo vode a simulácii efektu vodných prúdov.

Framework umožňuje experimentovanie s rôznymi prístupmi umelej inteligencie v prostredí Minecraftu. Môžu byť testované metódy navigácie za pomoci kopania tunelov či použitia štruktúr, ktoré si bot sám postaví. Zaujímavé taktiež môže byť použitie kooperácie viacerých botov spolu k plneniu náročných úloh, ako napríklad stavba veľkého mesta.

# Literatúra


[1] **Gemrot, J., Kadlec, R., Bida, M., Burkert, O., Pibil, R., Havlicek, J., Zemcak, L., Simlovic, J., Vansa, R., Stolba, M., Plch, T., Brom C.** (2009) : Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: Agents for Games and Simulations, LNCS 5920, Springer, pp. 1--15.

[2] **Engelbrecht, H. A., & Schiele, G.** (2014): Transforming Minecraft into a Research Platform. In: Proceedings of Consumer Communications and Networking Conference (CCNC). IEEE, pp. 257 – 262.

[3] **Vejman, M.** (2012): Pogamut and USARSim integration, Bc. Thesis, MFF, UK.

[4] **Budd, R.:** TorchBot, a new way to play. [Online]
[Dátum 22.7.2015] http://www.torchbot.net/

[5] **Gamberini, N.:** SpockBotMC [Online]
[Dátum 22.7.2015] http://github.com/SpockBotMC/SpockBot

[6] **wiki.vg:** Protocol [Online]
[Dátum 22.7.2015] http://wiki.vg/index.php?title=Protocol&oldid=5500

[7] **Wikipedia, The Free encyclopedia:** Minimum bounding box [Online]
[Dátum 22.7.2015]
https://en.wikipedia.org/wiki/Minimum_bounding_box#Axis-aligned_minimum_bounding_box

[8] **Hart, P. E.; Nilsson, N. J.; Raphael, B**. (1968): "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2): 100–107.

# Príloha 1 - CD

Na priloženom cd sa nachádzajú nasledujúce adresáre s uvedeným obsahom:

- *bin* – spustiteľný JAR súbor
- *javadoc* – vygenerovaná dokumentácia k programu
- *MinecraftBot* – zdrojové kódy k programu
- *text* – táto práca vo formáte PDF
- *tutorial* – návod na tvorbu vlastných botov
- *video* – video ukážky činnosti botov

# Príloha 2 – Užívateľská dokumentácia

V tejto prílohe sa nachádzajú základné informácie potrebné pre beh botov vytvorených v rámci tejto práce.

## Spustenie

Framework je dostupný ako samostatný JAR súbor v priečinku *bin*. Pre spustenie je potrebná Java verzie aspoň 1.8. Príkaz pre spustenie behu je:

```
java –jar MinecraftBot.jar
```

Po spustení sa textové rozhranie opýta číslo na bota, adresu a port serveru. Boti sa dokážu pripojiť na lokálne servery a dedikované servery s vypnutím nastavením *online-mode*. Server musí používať verziu hry 1.7.5. Hra by ideálne mala byť v survival móde a na peaceful obtiažnosti, inak nie je zaručená správna činnosť.

## Zoznam príkazov

Boti sa dajú ovládať pomocou príkazov zadávaných do herného chatu. Niektorí boti dokážu poslať zoznam svojich príkazov do chatu. Je však potreba brať ohľad na to, že server môže odpojiť botov z dôvodu spamovania, pokiaľ budú posielať správy do chatu príliš často.

### Responsive Bot

| | |
|---|---|
| `w` | – pohne sa dopredu o jedno políčko |
| `a` | – pohne sa doprava o jedno políčko |
| `s` | – pohne sa dozadu o jedno políčko |
| `d` | – pohne sa dopredu o jedno políčko |
| `j` | – vyskočí |
| `door` *x y z* | – použije dvere na pozícii *x, y, z* |
| `mine` | – vykope políčko pod sebou |
| `info` | – oznámi svoju pozíciu |
| `hello` | – pozdraví hráča a oznámi jeho pozíciu |
| `whatis` *x y z* | – popíše blok na pozícii *x, y, z* |
| `howmany` *id* | – oznámi počet predmetov s daným *id* v inventári |
| `navigate` | – pokúsi sa navigovať k hráčovi |
| `navigateto` *x y z* | – pokúsi sa navigovať pozíciu *x, y, z* |
| `respawn` | – oživí bota ak je mŕtvy |

## Builder

Builder dokáže stavať štruktúry načítané z textových súborov, ktoré sú uložené v priečinku *structures.*

`bl build` *`štruktúra modifikátor`*
– postaví štruktúru na pozícii hráča, môže byť modifikovaná
`bl structures` – vypíše zoznam načítaných štruktúr
`bl modifiers` – vypíše zoznam modifikátorov
`bl structures` – vypíše zoznam príkazov
`bl respawn` – oživí bota ak je mŕtvy
`bl help` – vypíše zoznam príkazov

Zoznam modifikátorov:
`flipx` – preklopí štruktúru cez osu X
`flipz` – preklopí štruktúru cez osu Z
`flipxz` – preklopí štruktúru cez osi X a Z (rotuje o 180 stupňov)
`rotatecw` – rotuje štruktúru o 90 stupňov v zápornom smere
`rotateccw` – rotuje štruktúru o 90 stupňov v kladnom smere

## Cleaner

Cleaner prenasleduje hráča a zbiera predmety spadnuté na zem v jeho okolí.

`cl followme` – začne prenasledovať hráča a zbierať predmety
`cl stop` – prestane prenasledovať hráča
`cl drop` – vyhodí všetky predmety z inventára
`cl respawn` – oživí bota ak je mŕtvy
`cl help` – vypíše zoznam príkazov

## Lumberjack

Lumberjack dostáva od hráčov úlohy na zbieranie dreva a plní ich.

`lj getwood` – pokúsi sa zohnať 12 drevených dosiek
`lj getwood` *x* – pokúsi sa *X* drevených dosiek
`lj givewood` – vyhodí všetko drevo z inventára
`lj giveall` – vyhodí všetky predmety z inventára
`lj comehere` – pokúsi sa navigovať k hráčovi
`lj stop` – ukončí aktuálnu úlohu
`lj respawn` – oživí bota ak je mŕtvy
`lj help` – vypíše zoznam príkazov